YB-60 Emulator Part 3

Landon
Lamoreaux
November 29th, 2023
CENG 325 – Project 3

The program is run exactly as specified in the program document.

1. Starting the program with an object file as an argument will load that into the emulator.
2. Typing in the address will print the data at that location.
3. Typing in 2 addresses with a "." between them will print all the data between those 2 locations.
4. Typing in an address with a ":" then bytes with spaces between them will replace anything after that address with the data entered in.
5. Typing in an address followed by an "r" will run all the code starting at that address.
6. Typing in an address followed by a "t" will disassemble the code and print out the instructions that they once were.
7. Typing in "info" will print out the contents of every register.
8. Typing in an address followed by an "s" will open a step through menu where the user can run one line of code at a time or print out the contents of the registers in-between lines of executed code.
    a. Typing in a "s" in the submenu will step to the next instruction.
    b. Typing in a "l" in the submenu will print out all the registers.
    c. Typing in a "q" in the submenu will quit the step through function.
9. You can exit the program by entering "exit", ctrl-C, or ctrl-D.

The program imports and uses the following packages:

1. numpy
2. sys
3. re
4. BitArray from bitstring

This emulator was developed in python 3.10.

If a user gives an address or data that is not in hexadecimal, then the program will output an error message and does not preform the function specified. It then prompts the user for new input.

If an error occurs that causes the program to fail, the error message: "Problem Encountered." Will be printed out. The program will quit what it was doing and return to the main menu.

There are 32, 32-bit registers, and 1048576 memory locations.

When a memory location followed by an "r" is entered the program counter is set to that memory address. The first 4 bytes are then grabbed and concatenated into the first instruction. That instruction is then parsed, and we determine the instruction format. Once it is parsed, we can print out all the bits in

each section of the instruction to the screen. We can also determine the label for the instruction and print that out as well. Then the instruction is executed and the program counter is incremented, then the next instruction is grabbed, parsed and printed until the ebreak instruction is encountered.

When a memory location followed by a “t” is entered the program counter is once again set to that memory address. The 4 bytes after that address are then concatenated and turned into a string for the instruction. The instruction string is then parsed to discover what each part of it means. We can then take the opcode, fnuct3, funct7, and imm and look up the instructions name. Next it prints out the full assembly instruction from what it was before it was turned into object code.

When a memory location followed by an “s” is entered, the program counter is again set to that memory address. The 4 bytes after that address are then concatenated and turned into a string for the instruction. The instruction string is then parsed to discover what each part of it means. We can then take the opcode, fnuct3, funct7, and imm and look up the instructions name. Next it prints out the full assembly instruction from what it was before it was turned into object code. It runs that line of code then prompts the user if they want to step to the next line of code, print out all the registers, or quit the execution.

Testing:

I tested the program by starting it with some of the provided object files, then running functions and comparing the output with the expected result from those instructions. My testing is detailed on the following pages.

This is the output I got from running the following disassembled code, this section runs a lot of the load and shift instructions, as well as several basic arithmetic instructions. It was posted by Alexander Gergen on November 28th. On the left side there is the disassembled code, in the center there is the output from running the disassembled code. Finally on the right side there is what remains in the registers after the instructions were executed.

```
> 300t
  addi x1, x0, 42
  addi x2, x0, -53
    lb x3, 764(x0)
    lh x4, 764(x0)
    lw x5, 764(x0)
   lbu x6, 764(x0)
   lhu x7, 764(x0)
  slli x8, x1, 2
   add x28, x2, x2
  slti x9, x2, 0
 sltiu x10, x2, 0
  xori x11, x2, 40
  srli x12, x5, 1
  srai x13, x5, 1027
   ori x14, x6, 151
  andi x15, x6, 150
 auipc x16, 47
   lui x17, 15
   add x18, x3, x1
   sub x19, x3, x1
  addi x31, x0, 2
   sll x20, x1, x31
   slt x21, x2, x0
  sltu x22, x2, x0
   xor x23, x31, x1
   srl x24, x2, x31
   sra x25, x2, x31
    or x26, x31, x1
   and x27, x31, x1
   beq x28, x2, 8
  addi x29, x0, 1
   bne x28, x0, 8
  addi x30, x0, 1
  addi x0, x0, 1
ebreak
```

```
> 300r
   PC      OPC   INST  rd    rs1  rs2/imm
00300 02A00093   ADDI 00001 00000 101010
00304 FCB00113   ADDI 00010 00000 111111001011
00308 2FC00183     LB 00011 00000 1011111100
0030C 2FC01203     LH 00100 00000 1011111100
00310 2FC02283     LW 00101 00000 1011111100
00314 2FC04303    LBU 00110 00000 1011111100
00318 2FC05383    LHU 00111 00000 1011111100
0031C 00209413   SLLI 01000 00001 00010
00320 00210E33    ADD 11100 00010 00010 00000
00324 00012493   SLTI 01001 00010 00000
00328 00013513  SLTIU 01010 00010 00000
0032C 02814593   XORI 01011 00010 101000
00330 0012D613   SRLI 01100 00101 00001
00334 4032D693   SRAI 01101 00101 10000000011
00338 09736713    ORI 01110 00110 10010111
0033C 09637793   ANDI 01111 00110 10010110
00340 0002F817  AUIPC 10000       101111
00344 0000F8B7    LUI 10001       01111
00348 00118933    ADD 10010 00011 00001 00000
0034C 401189B3    SUB 10011 00011 00001 00000
00350 00200F93   ADDI 11111 00000 00010
00354 01F09A33    SLL 10100 00001 11111 00000
00358 00012AB3    SLT 10101 00010 00000 00000
0035C 00013B33   SLTU 10110 00010 00000 00000
00360 001FCBB3    XOR 10111 11111 00001 00000
00364 01F15C33    SRL 11000 00010 11111 00000
00368 41F15CB3    SRA 11001 00010 11111 00000
0036C 001FED33     OR 11010 11111 00001 00000
00370 001FFDB3    AND 11011 11111 00001 00000
00374 002E0463    BEQ       11100 00010 01000
00378 00100E93   ADDI 11101 00000 00001
0037C 000E1463    BNE       11100 00000 01000
00384 00100013   ADDI 00000 00000 00001
00388 00100073 EBREAK
```

```
> info
 x0 00000000
 x1 0000002A
 x2 FFFFFFCB
 x3 FFFFFF8F
 x4 FFFFCD8F
 x5 8142CD8F
 x6 0000008F
 x7 0000CD8F
 x8 000000A8
 x9 00000001
x10 00000000
x11 FFFFFFE3
x12 40A166C7
x13 F02859B1
x14 0000009F
x15 00000086
x16 0002F340
x17 0000F000
x18 FFFFFFB9
x19 FFFFFF65
x20 000000A8
x21 00000001
x22 00000000
x23 00000028
x24 3FFFFFF2
x25 FFFFFFF2
x26 0000002A
x27 00000002
x28 FFFFFF96
x29 00000001
x30 00000000
x31 00000002
```

This is the output I got from running the following disassembled code, this section runs most of the branch and extended instructions. It was posted by Alexander Gergen on November 29th. On the left side there is the disassembled code, in the center there is the output from running the disassembled code. Finally on the right side there is what remains in the registers after the instructions were executed.

```
> 300t
  addi x1, x0, 127
   lhu x2, 762(x0)
    lw x3, 764(x0)
    sb x1, 761(x0)
    sh x2, 762(x0)
    sw x3, 764(x0)
  addi x4, x0, -43
   blt x4, x0, 8
  addi x5, x0, 7
  bltu x4, x0, 8
    lw x6, 756(x0)
   bge x4, x0, 8
    lw x7, 752(x0)
  bgeu x4, x0, 8
  addi x8, x0, -1107
   jal x9, 8
  jalr x10, 840(x0)
   jal x0, -4
  addi x11, x0, 3
   mul x12, x11, x5
  mulh x13, x2, x6
mulhsu x14, x4, x6
 mulhu x15, x2, x6
   div x16, x2, x4
  divu x17, x2, x4
   rem x18, x2, x4
  remu x19, x2, x4
ebreak
```

```
> 300r
   PC      OPC   INST  rd    rs1  rs2/imm
00300 07F00093   ADDI 00001 00000 1111111
00304 2FA05103    LHU 00010 00000 1011111010
00308 2FC02183     LW 00011 00000 1011111100
0030C 2E100CA3     SB       00000 00001 1011111001
00310 2E201D23     SH       00000 00010 1011111010
00314 2E302E23     SW       00000 00011 1011111100
00318 FD500213   ADDI 00100 00000 111111010101
0031C 00024463    BLT       00100 00000 01000
00324 00026463   BLTU       00100 00000 01000
00328 2F402303     LW 00110 00000 1011110100
0032C 00025463    BGE       00100 00000 01000
00330 2F002383     LW 00111 00000 1011110000
00334 00027463   BGEU       00100 00000 01000
0033C 008004EF    JAL 01001       01000
00344 FFDFF06F    JAL 00000       11111111111111111100
00340 34800567   JALR 01010 00000 1101001000
00348 00300593   ADDI 01011 00000 00011
0034C 02558633    MUL 01100 01011 00101 00000
00350 026116B3   MULH 01101 00010 00110 00000
00354 02622733 MULHSU 01110 00100 00110 00000
00358 026137B3  MULHU 01111 00010 00110 00000
0035C 02414833    DIV 10000 00010 00100 00000
00360 024158B3   DIVU 10001 00010 00100 00000
00364 02416933    REM 10010 00010 00100 00000
00368 024179B3   REMU 10011 00010 00100 00000
0036C 00100073 EBREAK
```

```
> info
 x0 00000000
 x1 0000007F
 x2 00007F00
 x3 A9AEC27B
 x4 FFFFFFD5
 x5 00000000
 x6 00C0FFEE
 x7 000FADED
 x8 00000000
 x9 00000340
x10 00000344
x11 00000003
x12 00000000
x13 0000005F
x14 FFFFFFFF
x15 0000005F
x16 FFFFFD0B
x17 00000000
x18 FFFFFFD9
x19 00007F00
x20 00000000
x21 00000000
x22 00000000
x23 00000000
x24 00000000
x25 00000000
x26 00000000
x27 00000000
x28 00000000
x29 00000000
x30 00000000
x31 00000000
```

This is the output I got from running the following disassembled code, this section tests some branching and I type instructions. The object code was posted by Tim Bilik on November 27th, this is the top bit of code. On the left side there is the disassembled code, in the center there is the output from running the disassembled code. Finally on the right side there is what remains in the registers after the instructions were executed.

```
> 300t
   lui x5, 60800
  addi x5, x5, 1202
    sb x5, 1024(x0)
    sh x5, 1028(x0)
    sw x5, 1032(x0)
    lb x28, 1024(x0)
    lh x29, 1028(x0)
    lw x30, 1032(x0)
   lbu x31, 1024(x0)
   lhu x9, 1028(x0)
  slti x18, x0, -1
   lui x20, 300
 sltiu x19, x20, -1
  xori x21, x19, 3
   ori x22, x21, 4
  andi x23, x22, 4
  slli x24, x23, 2
  addi x25, x0, -4
  srli x26, x25, 1
  srai x27, x25, 1025
ebreak
```

```
> 300r
   PC      OPC   INST  rd    rs1  rs2/imm
00300 0ED802B7    LUI 00101       1110110110000000
00304 4B228293   ADDI 00101 00101 10010110010
00308 40500023     SB       00000 00101 10000000000
0030C 40501223     SH       00000 00101 10000000100
00310 40502423     SW       00000 00101 10000001000
00314 40000E03     LB 11100 00000 10000000000
00318 40401E83     LH 11101 00000 10000000100
0031C 40802F03     LW 11110 00000 10000001000
00320 40004F83    LBU 11111 00000 10000000000
00324 40405483    LHU 01001 00000 10000000100
00328 FFF02913   SLTI 10010 00000 111111111111
0032C 0012CA37    LUI 10100       100101100
00330 FFFA3993  SLTIU 10011 10100 111111111111
00334 0039CA93   XORI 10101 10011 00011
00338 004AEB13    ORI 10110 10101 00100
0033C 004B7B93   ANDI 10111 10110 00100
00340 002B9C13   SLLI 11000 10111 00010
00344 FFC00C93   ADDI 11001 00000 111111111100
00348 001CDD13   SRLI 11010 11001 00001
0034C 401CDD93   SRAI 11011 11001 10000000001
00350 00100073 EBREAK
```

```
> info
 x0 00000000
 x1 00000000
 x2 00000000
 x3 00000000
 x4 00000000
 x5 0ED804B2
 x6 00000000
 x7 00000000
 x8 00000000
 x9 000004B2
x10 00000000
x11 00000000
x12 00000000
x13 00000000
x14 00000000
x15 00000000
x16 00000000
x17 00000000
x18 00000000
x19 00000001
x20 0012C000
x21 00000002
x22 00000006
x23 00000004
x24 00000010
x25 FFFFFFFC
x26 7FFFFFFE
x27 FFFFFFFE
x28 FFFFFFB2
x29 000004B2
x30 0ED804B2
x31 000000B2
```